libEnsemble: A Library for Managing Dynamic Ensembles of Calculations

David Bindel Stephen Hudson John-Luke Navarro
Jeffrey Larson Stefan Wild

Argonne National Laboratory

July 14, 2020

What is `libEnsemble`

- `libEnsemble` is a library to coordinate the concurrent evaluation of dynamic ensembles of calculations

What is `libEnsemble`

- `libEnsemble` is a library to coordinate the concurrent evaluation of dynamic ensembles of calculations

- Developed to use massively parallel resources to accelerate the solution of design, decision, and inference problems

What is libEnsemble

- libEnsemble is a library to coordinate the concurrent evaluation of dynamic ensembles of calculations
- Developed to use massively parallel resources to accelerate the solution of design, decision, and inference problems
- Developed to expand the class of problems that can benefit from increased computational concurrency levels

What is `libEnsemble`

- `libEnsemble` is a library to coordinate the concurrent evaluation of dynamic ensembles of calculations

- Developed to use massively parallel resources to accelerate the solution of design, decision, and inference problems

- Developed to expand the class of problems that can benefit from increased computational concurrency levels

- `libEnsemble` uses a manager to allocate work to various workers

What is `libEnsemble`

- `libEnsemble` is a library to coordinate the concurrent evaluation of dynamic ensembles of calculations

- Developed to use massively parallel resources to accelerate the solution of design, decision, and inference problems

- Developed to expand the class of problems that can benefit from increased computational concurrency levels

- `libEnsemble` uses a manager to allocate work to various workers

- A `libEnsemble` worker is the smallest indivisible unit to perform

libEnsemble requires of the user

gen_f: Generates inputs to sim_f and alloc_f

libEnsemble requires of the user

gen_f: Generates inputs to sim_f and alloc_f

sim_f: Evaluates a simulation (i.e., user-defined function) using input defined by gen_f

`libEnsemble` requires of the user

`gen_f`: Generates inputs to `sim_f` and `alloc_f`

`sim_f`: Evaluates a simulation (i.e., user-defined function) using input defined by `gen_f`

`alloc_f`: Decides whether (or not) `sim_f` or `gen_f` should be called (and with what input/resources) as workers become available

libEnsemble dependencies

- Python 3.5+, NumPy, psutil

libEnsemble dependencies

- Python 3.5+, NumPy, psutil

If using MPI communications in libEnsemble

- An MPI implementation (e.g., MPICH) built with shared/dynamic libraries
- mpi4py v2.0.0 or above

libEnsemble dependencies

- Python 3.5+, NumPy, psutil

If using MPI communications in libEnsemble

- An MPI implementation (e.g., MPICH) built with shared/dynamic libraries

- mpi4py v2.0.0 or above

- Can also use multiprocessing or TCP for libEnsemble communications

`libEnsemble` dependencies

- Python 3.5+, NumPy, psutil

If using MPI communications in `libEnsemble`

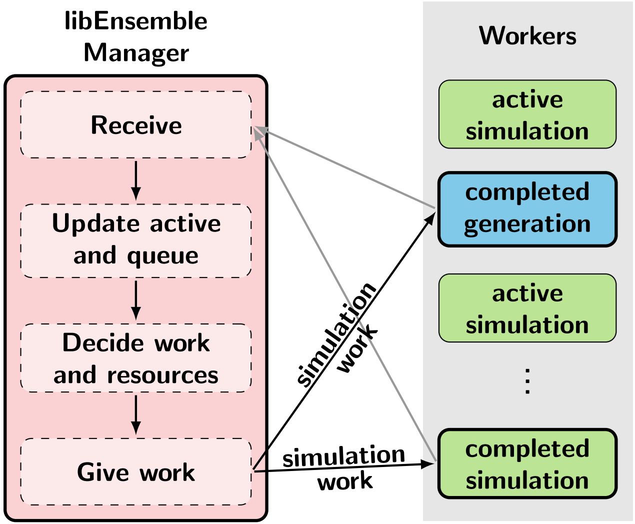
- An MPI implementation (e.g., MPICH) built with shared/dynamic libraries

- mpi4py v2.0.0 or above

- Can also use multiprocessing or TCP for `libEnsemble` communications

- Example `gen_f`/`sim_f` functions require NLopt, PETSc, SciPy, Tasmanian, etc.

`libEnsemble` overview



libEnsemble Manager

Receive

Update active and queue

Decide work and resources

Give work

Workers

active simulation

completed generation

active simulation

⋮

completed simulation

Possible user requirements of `libEnsemble`

- `sim_f/gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations

Possible user requirements of `libEnsemble`

- `sim_f/gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation
- Ability to recover hardware resources from `libEnsemble`-/generation-/simulation-terminated calculations

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation
- Ability to recover hardware resources from `libEnsemble`-/generation-/simulation-terminated calculations
- Simulation/generation checkpoint and restart

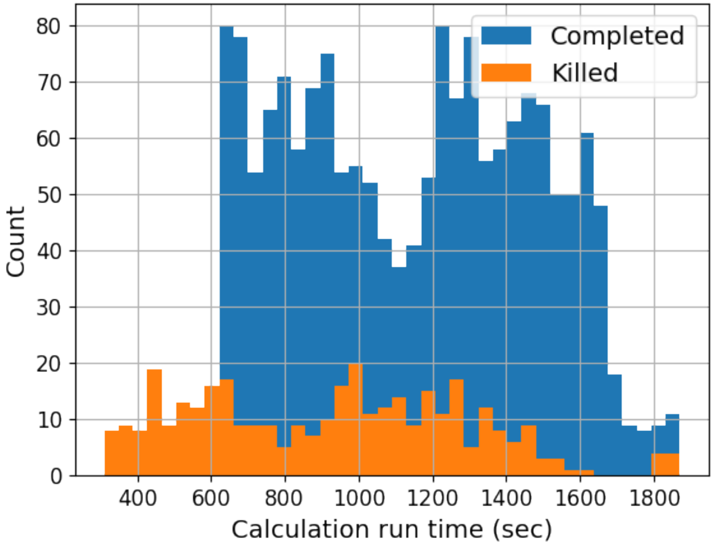
Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation
- Ability to recover hardware resources from `libEnsemble`-/generation-/simulation-terminated calculations
- Simulation/generation checkpoint and restart
- Execution on multiple LCFs (Summit/OLCF, Cori/NERSC, Theta/ALCF)

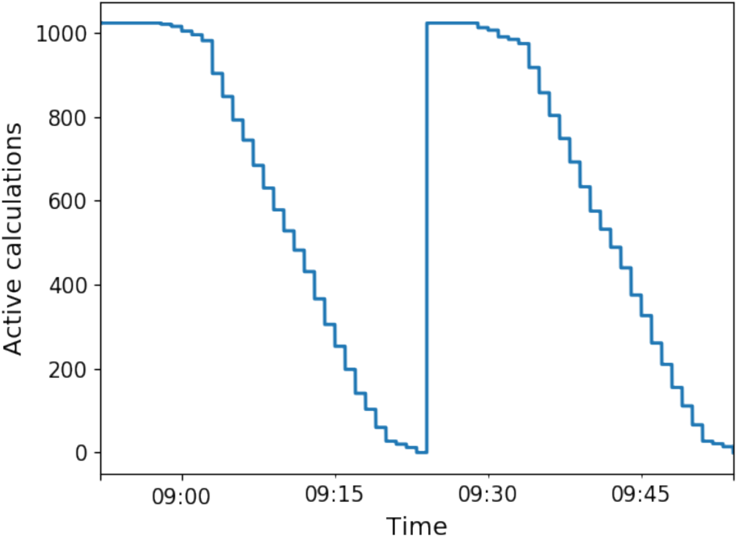
Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation
- Ability to recover hardware resources from `libEnsemble`-/generation-/simulation-terminated calculations
- Simulation/generation checkpoint and restart
- Execution on multiple LCFs (Summit/OLCF, Cori/NERSC, Theta/ALCF)
- Thousands of concurrent workers

OPAL particle accelerator simulations on 1024 Theta nodes



OPAL particle accelerator simulations on 1024 Theta nodes



Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel

- Add another level of parallelism to a simulation that no longer scales

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel

- Add another level of parallelism to a simulation that no longer scales

- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel
- Add another level of parallelism to a simulation that no longer scales
- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work
- Don't have to write your own kills, just complete `libEnsemble` templates

Why would you want to use `libEnsemble`?

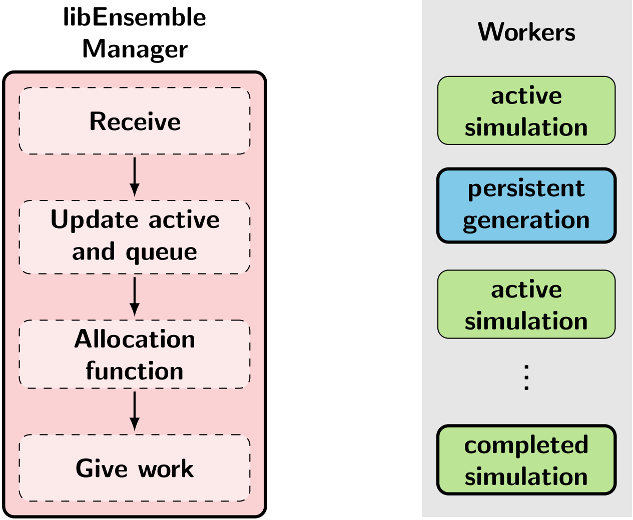
- Easily take serial code and start running in parallel
- Add another level of parallelism to a simulation that no longer scales
- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work
- Don't have to write your own kills, just complete `libEnsemble` templates
- Want to add concurrency to a generator (e.g., multiple local optimizers.)

Why `libEnsemble` and not. . . ?

Swift: (the parallel scripting language)

- “Can run million programs, thousands at a time, launching hundreds per second”
- Require writing your generators in Swift’s scripting language
- Difficult to tightly couple generation of inputs and future/active running simulations

libEnsemble overview


libEnsemble Manager

Receive

Update active and queue

Allocation function

Give work

Workers

active simulation

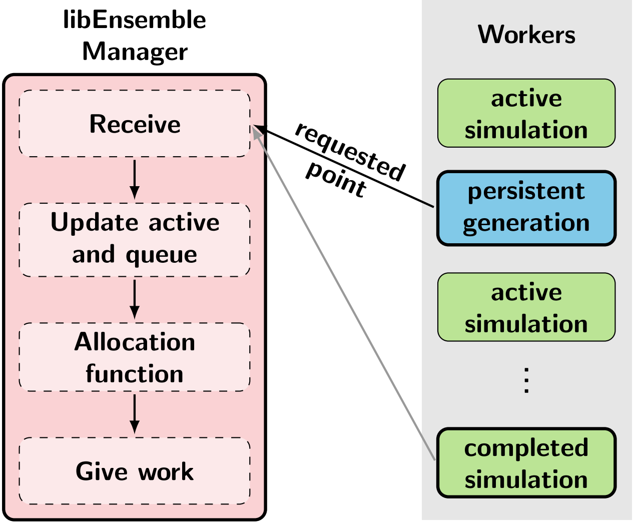
persistent generation

active simulation

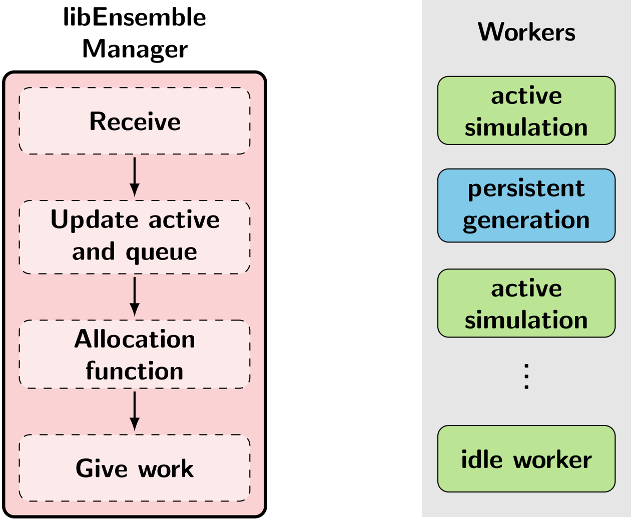
⋮

completed simulation

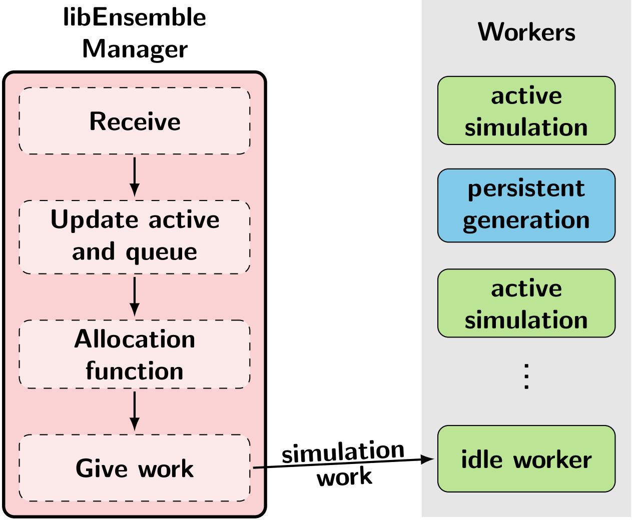
libEnsemble overview



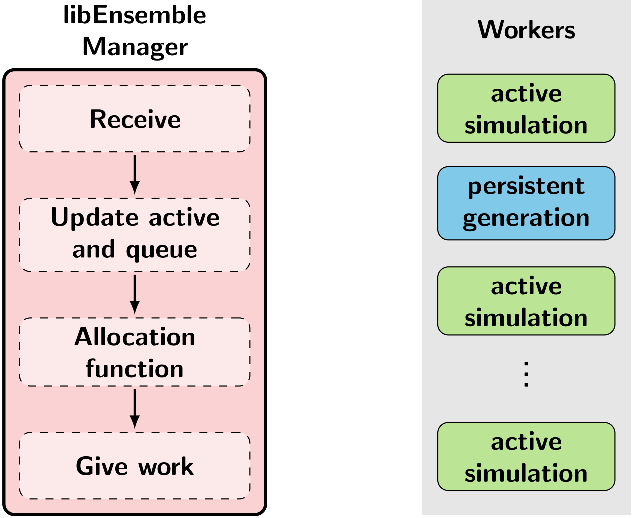
`libEnsemble` overview



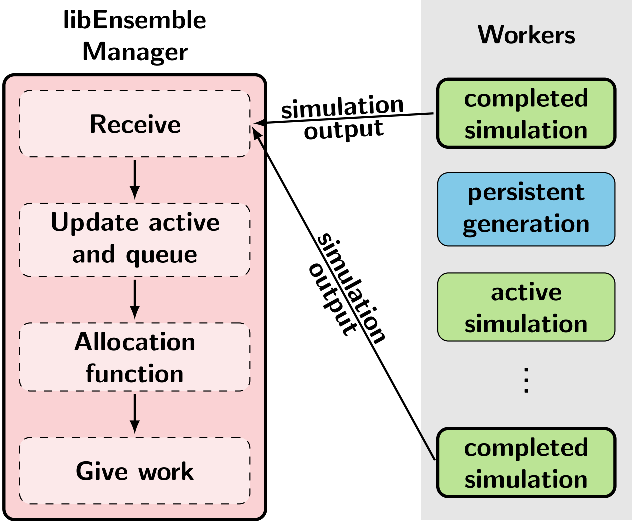
libEnsemble overview



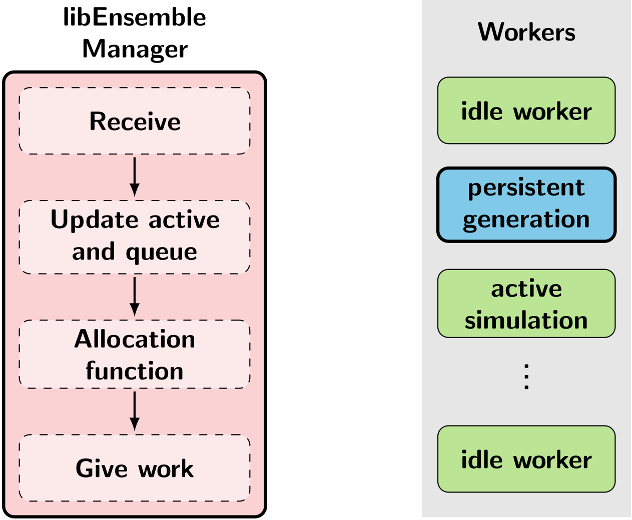
libEnsemble overview



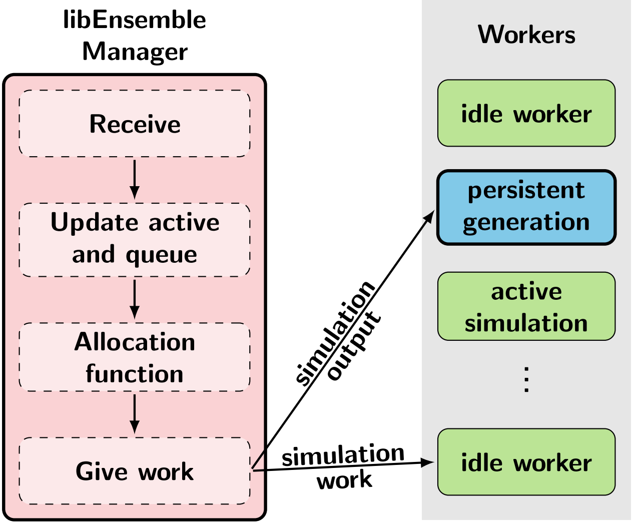
libEnsemble overview



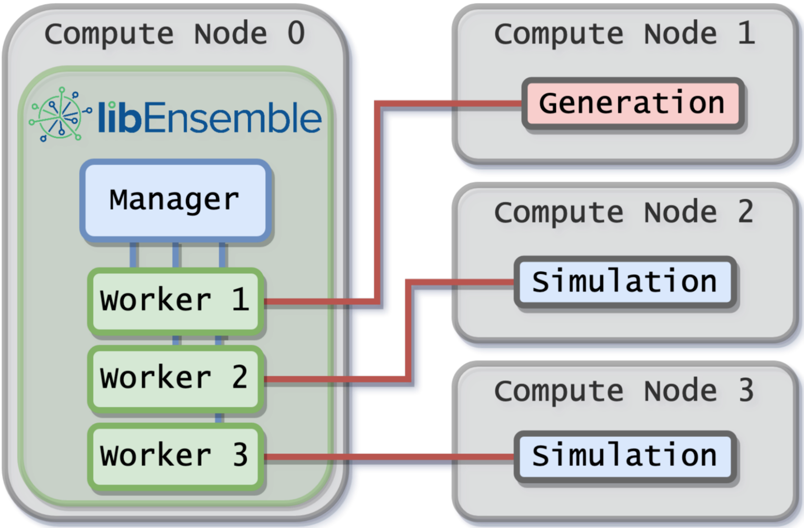
libEnsemble overview



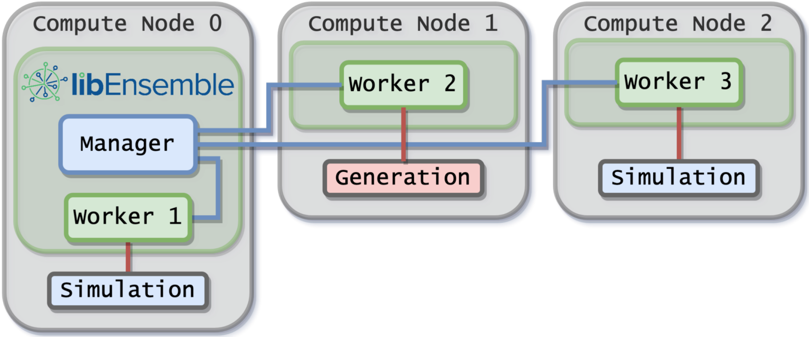
libEnsemble overview



libEnsemble modes: centralized or distributed



libEnsemble modes: centralized or distributed



Timing `libEnsemble` overhead

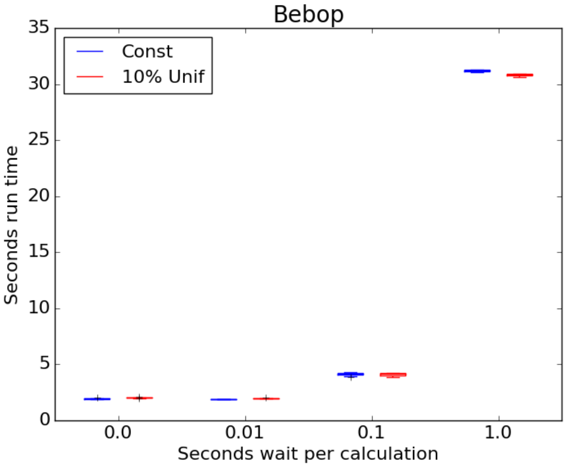
- Time for `libEnsemble` to sample/evaluation 30×(workers) points

Timing libEnsemble overhead

- Time for libEnsemble to sample/evaluation 30×(workers) points
- Tons of system-dependent caveats

Timing `libEnsemble` overhead

- Time for `libEnsemble` to sample/evaluation 30×(workers) points
- Tons of system-dependent caveats
- 32 nodes × 36 cores = 1152-1 workers



Use cases

- A user wants to optimize a function that depends on a simulation
- The simulation is already using parallel resources, but not a large fraction of some computer
- `libEnsemble` can coordinate the concurrent evaluation of the simulation `sim_f` at various parameter values and `gen_f` would return candidate parameter values (possibly after each `sim_f` output)

Use cases

- A user has a `gen_f` that produces different meshes to be used within a `sim_f`
- Given the `sim_f` output, `gen_f` will refine a mesh or produce a new mesh
- `libEnsemble` can ensure that the calculated meshes can be used by multiple simulations without requiring movement of data

Use cases

- A user wants to evaluate a simulation `sim_f` at parameters sampled from a set of parameter values
- Many parameter sets will cause the simulation to fail
- `libEnsemble` can stop unresponsive evaluations, and recover computational resources for future evaluations
- `gen_f` can update the sampling after discovering regions where evaluations of `sim_f` fail

Use cases

- A user has a simulation `sim_f` that requires calculating multiple expensive quantities, some of which depend on other quantities
- `libEnsemble` can observe intermediate quantities in order to stop related calculations and preempt future calculations associated with a poor parameter values

Use cases

- A user wishes to identify multiple local optima for a `sim_f`
- `libEnsemble` can use the points from the APOSMM `gen_f` to identify optima

Use cases

- A user wishes to identify multiple local optima for a `sim_f`
- `libEnsemble` can use the points from the APOSMM `gen_f` to identify optima

Naturally, combinations of use cases is supported as well

Problem setup

- We want to identify distinct, “high-quality”, local minimizers of

$$\begin{aligned} \text{minimize}\, f(x) \\ l \leq x \leq u \\ x \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective

Problem setup

- We want to identify distinct, “high-quality”, local minimizers of

$$\begin{aligned} \text{minimize}\, f(x) \\ l \leq x \leq u \\ x \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective
- Derivatives of f may or may not be available

Problem setup

- We want to identify distinct, “high-quality”, local minimizers of

$$\begin{aligned} \text{minimize}\, f(x) \\ l \le x \le u \\ x \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective
- Derivatives of f may or may not be available
- The simulation f is likely using parallel resources, but it does not utilize the entire machine

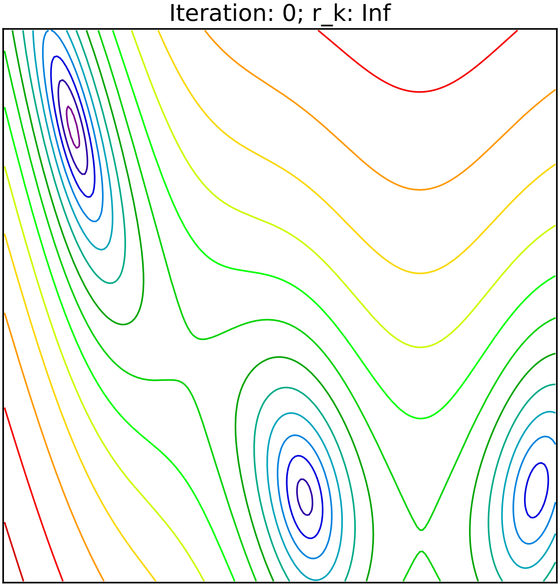
Problem setup

- We want to identify distinct, “high-quality”, local minimizers of

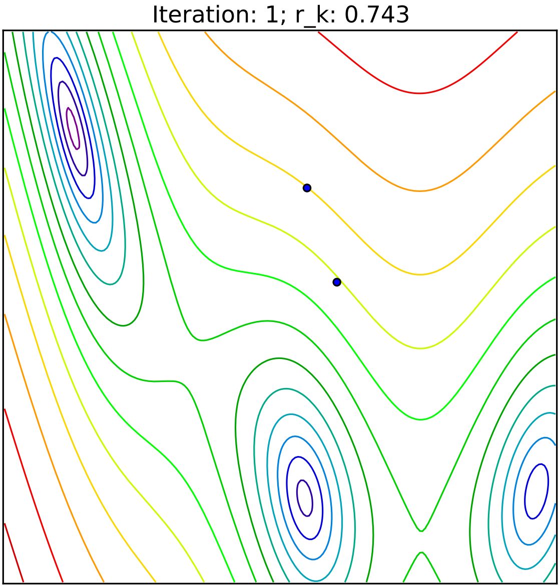
$$\begin{aligned} \text{minimize}\, f(x) \\ l \leq x \leq u \\ x \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective
- Derivatives of f may or may not be available
- The simulation f is likely using parallel resources, but it does not utilize the entire machine
- Possibly have a specialized local optimization method for f

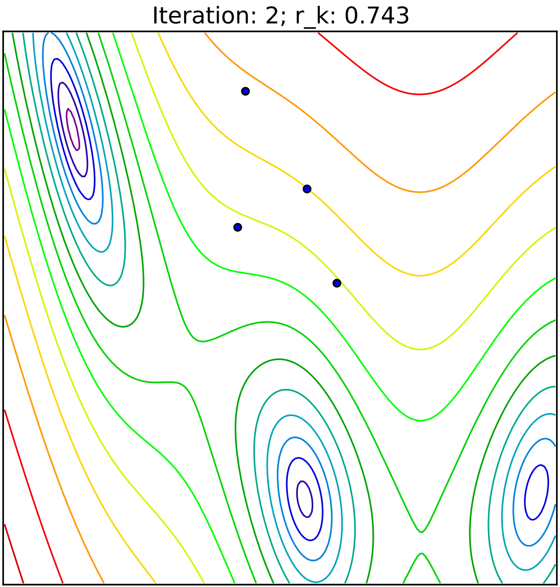
APOSMM



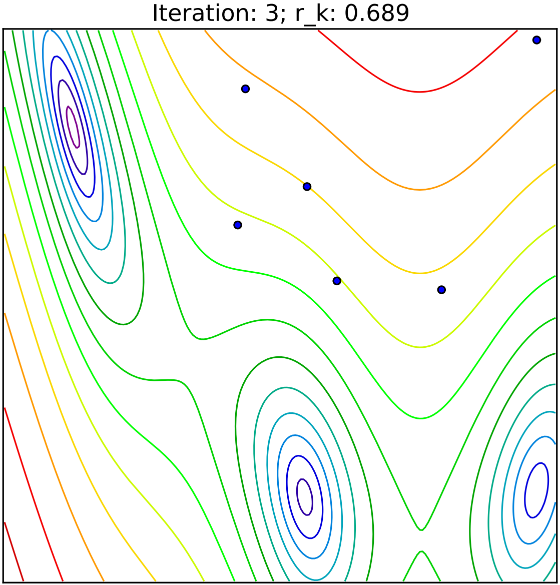
APOSMM



APOSMM

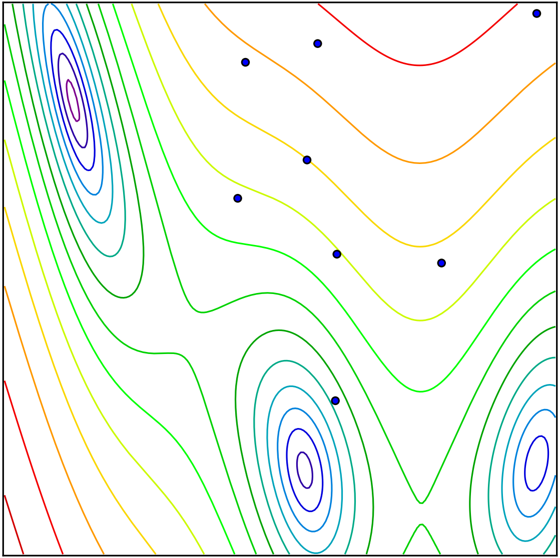


APOSMM

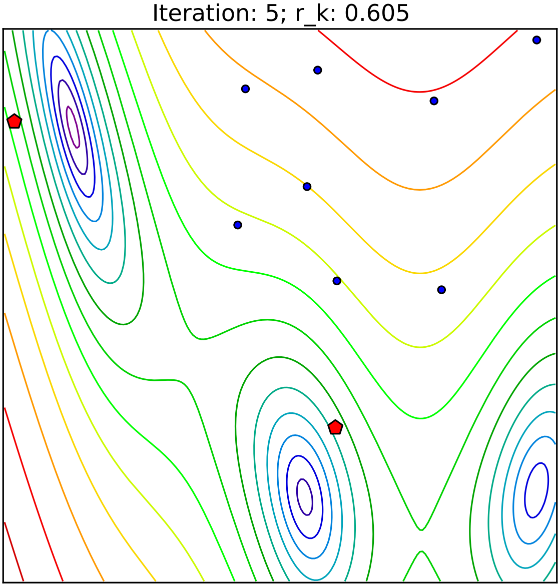


APOSMM

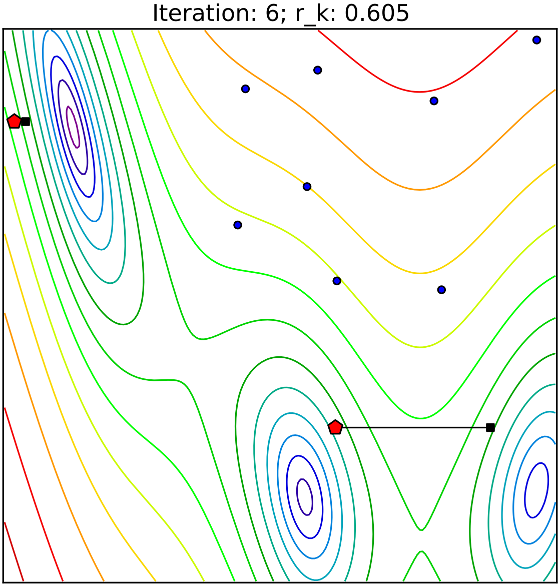
Iteration: 4; r_k: 0.643



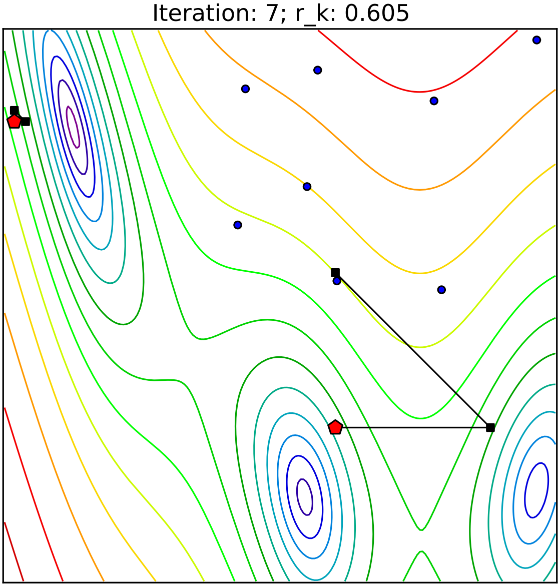
APOSMM



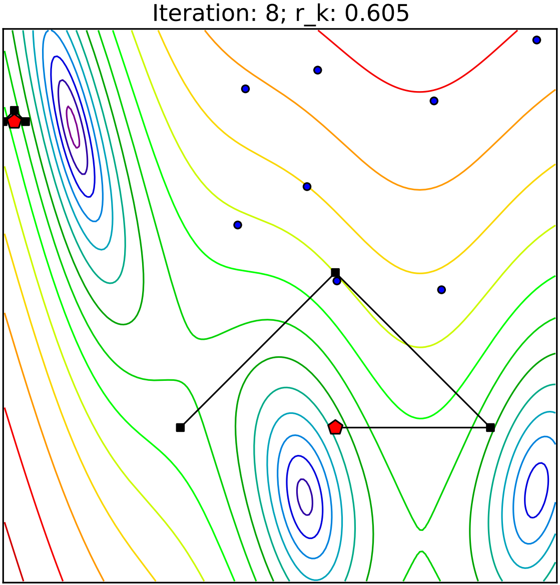
APOSMM



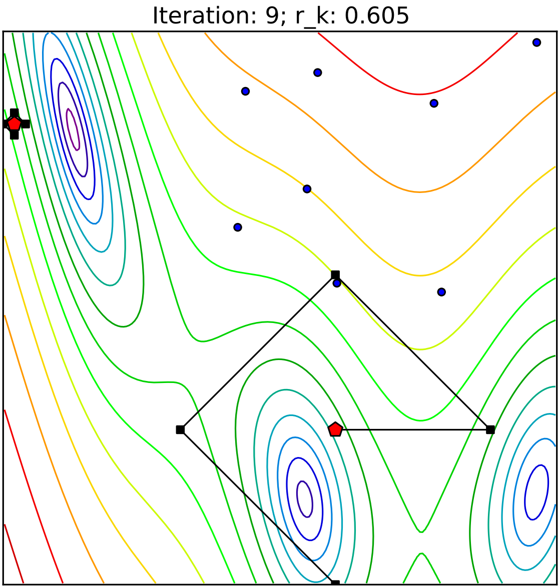
APOSMM



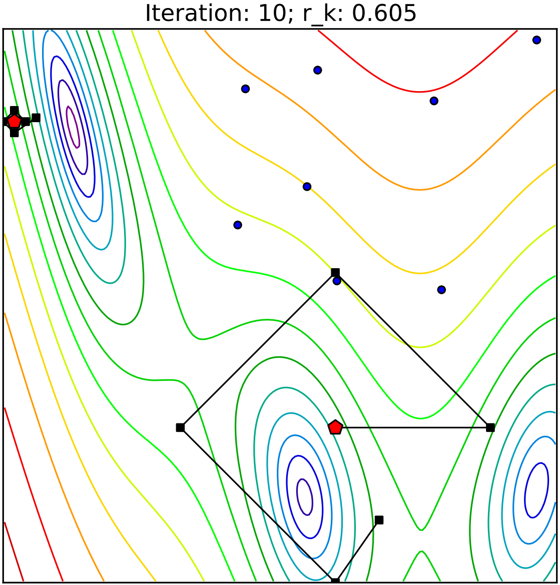
APOSMM



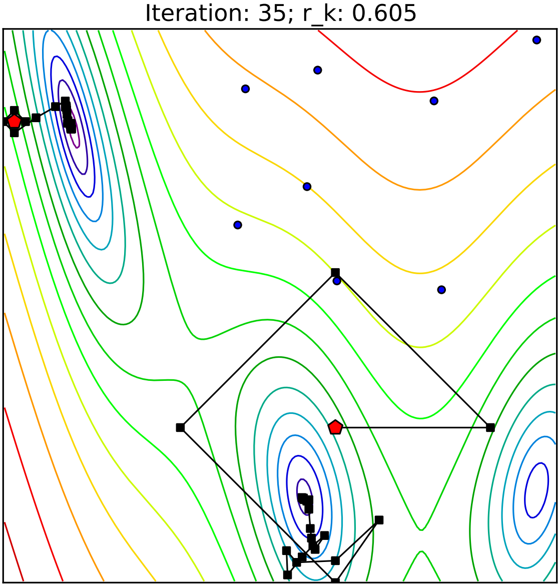
APOSMM



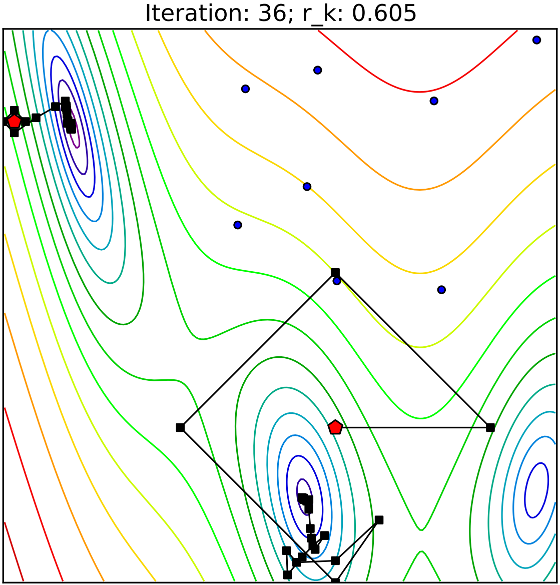
APOSMM



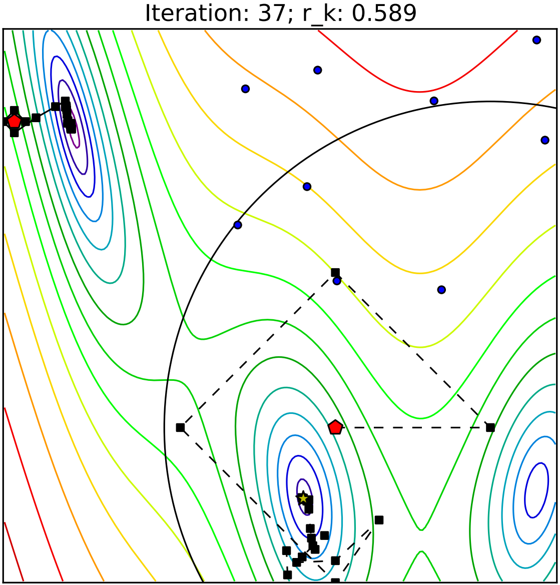
APOSMM



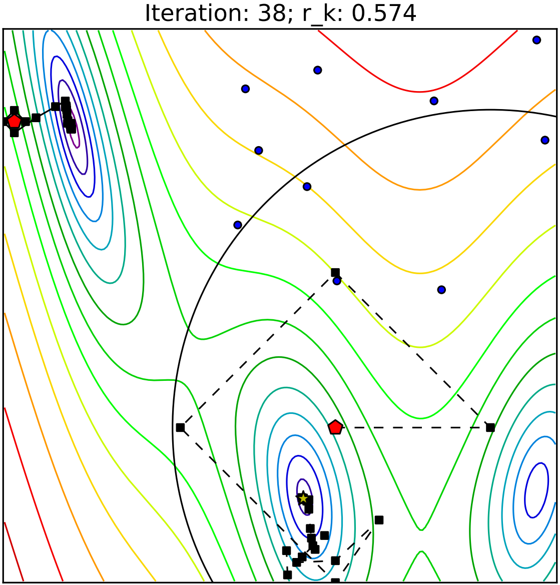
APOSMM



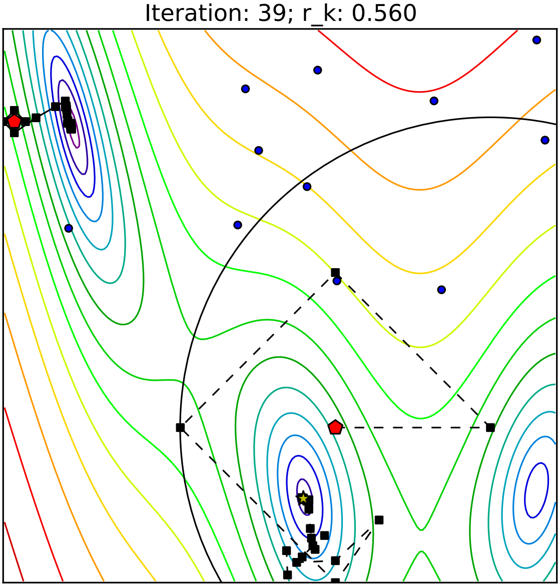
APOSMM



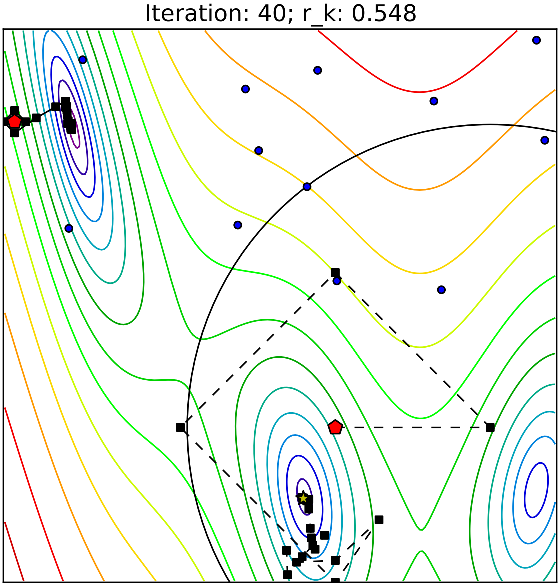
APOSMM



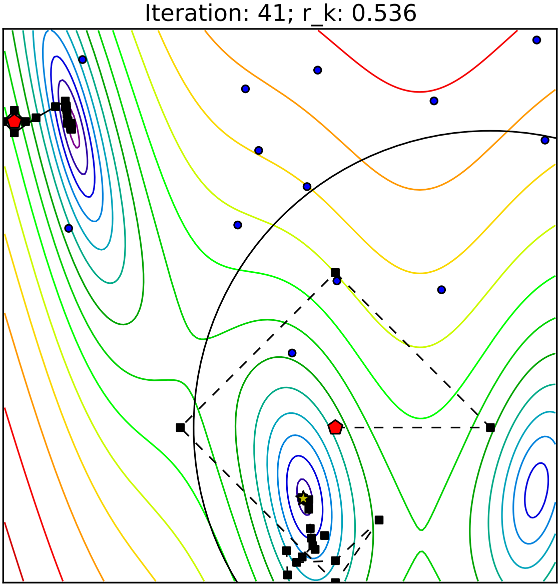
APOSMM



APOSMM

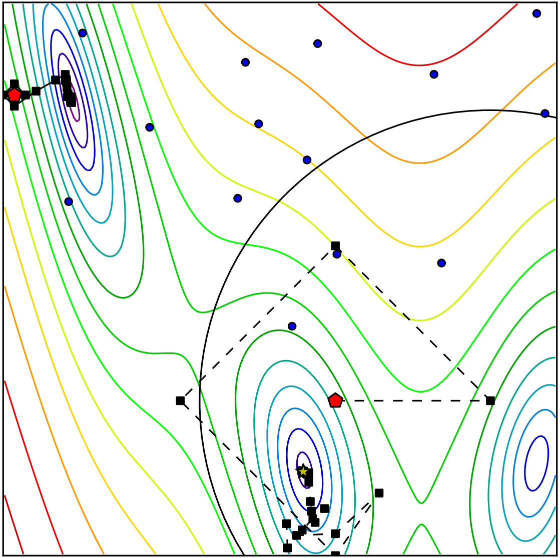


APOSMM

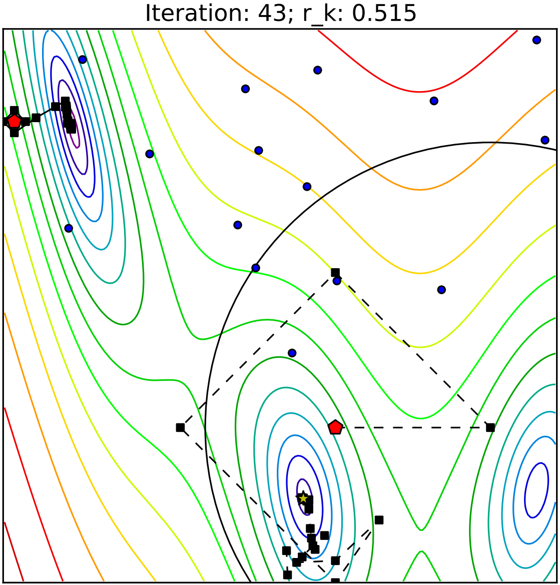


APOSMM

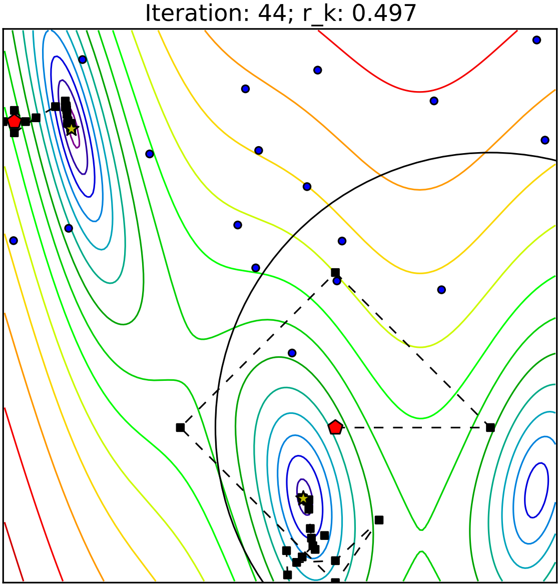
Iteration: 42; r_k: 0.525



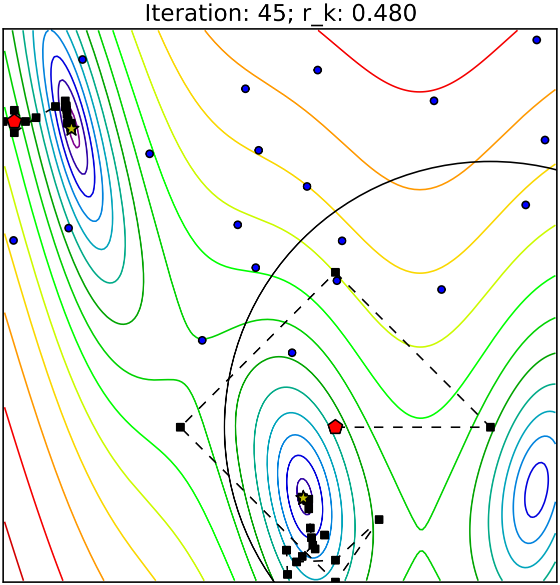
APOSMM



APOSMM



APOSMM



Closing Remarks

- We have a growing set of use cases and examples
- Let us know if you have examples you'd like to see
- `https://github.com/Libensemble/libensemble`